Performance Prediction: A case study using a multi-ring KSR-1 machine[*]

Xian-He Sun

Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803-4020

Jianping Zhu

NSF Engineering Research Center
Dept. of Math. and Stat.
Mississippi State University
Mississippi State, MS 39762

Abstract

As computers with tens of thousands of processors are successfully delivering high performance power for solving some of the so-called "grand-challenge" applications, the notion of scalability is becoming an important metric in the evaluation of parallel machine architectures and algorithms. In this study, the prediction of scalability and its application are carefully investigated. A simple formula is presented to show the relation between scalability, single processor computing power, and degradation of parallelism. A case study is conducted on a multi-ring KSR-1 shared virtual memory machine. Experimental and theoretical results show that the influence of topology variation of an architecture is predictable. Therefore, the performance of an algorithm on a sophisticated, hierarchical architecture can be predicted and the best algorithm-machine combination can be selected for a given application.

[*]This research was supported in part by the National Aeronautics and Space Administration under NASA contract NAS1-19480 while the first author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001 and was supported in part by NASA contract NAS1-1672/MMP.

1 Introduction

With modern technology, parallel processing seems to be the only way to achieve higher performance. In recent years, various architectures have been proposed to connect a large number of processors into a single powerful machine; and various algorithms have been developed on these machines to explore the potential of high computation power. However, each architecture has distinct properties, and each algorithm has its own inherent data structures. The performance of an algorithm on a particular architecture may vary significantly as the system and problem sizes increase. Predicting the performance of an algorithm-machine combination is difficult and elusive.

There are two commonly used synchronization and communication models: message-passing and shared-memory. Processes communicate through explicit message passing in the message-passing model and through shared variables in the shared-memory model. Traditionally, message-passing is the natural choice of distributed-memory machines. With shared virtual address space, shared virtual memory can be supported on distributed-memory machines, but requires sophisticated hardware and system support. Shared virtual memory machines combine the merits of both the distributed-memory machines and the shared-memory communication model. They are scalable and provide sequential-like programming environment. However, performance prediction of shared virtual memory machines is more difficult than that of traditional message-passing machines, because their communication is implicit and memory access time is non-uniform.

Simply speaking, a scalable architecture is an architecture capable of yielding very high raw computation power when the system size is large. However the high computation power may not be realized in solving a given application, since the achievable efficiency of an application may drop quickly with the increase of system size. To evaluate the ability of maintaining performance, several metrics have been proposed to measure the scalability of algorithm-machine combinations [2, 3, 7, 8, 11]. Isospeed scalability [8] is one of the proposed metrics. It measures the ability of an algorithm-machine combination to maintain unit processor speed. Through a case study in this paper, we investigate issues of performance prediction of shared virtual memory machines. Performance models are developed in terms of execution time and scalability. Experimental results on a 64-node Kendall Square KSR-1 show that, when performance information of small scale systems is available, the performance of large scale systems can be predicted. Thus, machine architectures and algorithms can be compared in terms of scalability without run-time information. Since a 64- node KSR- 1 is a shared virtual memory machine with variable memory access times, the experience learned in this study is reasonably general and should extend to a class of applications.

The paper is organized as follows. In section 2, we first review isospeed scalability, then the properties of isospeed scalability are discussed. Performance formulas also developed to show the relations between execution time and scalability and to show possible approaches of predicting scalability. In Section 3, the regularized least squares application and the KSR-1 architecture are introduced. Theoretical analysis is given to find the performance bound of the application and to develop the performance model of the algorithm-machine combination. Experimental details and results, which match the predicted performance closely, are given in Section 4. A practical method is introduced to measure the memory access delay and other system overhead of the multi-level ring, shared virtual memory machine. Performance prediction without run-time information and selection of an appropriate algorithm-architecture combination for a given application are also discussed in Section 4. Finally, the summary is given in Section 5.

2 Definition and Analysis

One of the main motivations of parallel processing is to solve large problems fast. Considering both execution time and problem size, what we seek from parallel processing is speed which is defined as work divided by time. In general, how work should be defined is controversial. For scientific applications, it is commonly agreed that the floating point (flop) operation count is a good estimate of work (problem size)[1]. The average unit speed (or average speed, in short) is a good measure of parallel processing speed.

Definition 1 *The* `average unit speed` *is the achieved speed of the given computing system divided by p, the number of processors.*

In the ideal situation, average speed remains constant when system size increases. Hardware peak performance provided by vendors is usually based on this ideal assumption. If problem size is fixed, the ideal situation is unlikely to happen in practice, since when problem size is fixed, the communication/computation ratio is likely to increase with the number of processors, and therefore, the speed will decrease with increased system size. On the other hand, if system size is fixed, communication/computation ratio is likely to decrease with increased problem size for most practical algorithms. For these algorithms, increasing problem size with the system size may keep the average speed constant. Based on this observation, *the isospeed scalability* has been formally defined as the ability to maintain the average speed in [8].

Definition 2 *An* `algorithm-machine combination` *is* `scalable` *if the achieved average speed of the algorithm on the given machine can remain constant with increasing numbers of processors, provided the problem size can be increased with the system size.*

For a large class of algorithm-machine combinations, the average speed can be maintained by increasing problem size [8]. The necessary increase of problem size varies with algorithms, machines,

[1]Some authors refer to problem size as the parameter that determines the work, for instance, the order of matrices. In this paper, problem size refers to the work to be performed and we will use problem size and work alternatively.

and their combinations. This variation provides a quantitative measurement for scalability. Let W be the amount of work of an algorithm when p processors are employed in a machine, and let W' be the amount of work needed to maintain the average speed when $p' > p$ processors are employed. Then we define the *scalability from system size p to system size p'* of the algorithm-machine combination as follows:

$$\psi(p, p') = \frac{p'W}{pW'} \tag{1}$$

The work W' is determined by the isospeed constraint. When $W' = \frac{p'}{p}W$, that is when average speed is maintained with work per processor unchanged, the scalability equals one. It is the ideal case. In general, work per processor may have to be increased to achieve the fixed average speed, and scalability is less than one.

Speedup is a widely used performance metric in parallel processing. It is defined as sequential execution time over parallel execution time and is used to measure the parallel processing gain over sequential processing. Traditionally, parallel efficiency is defined as speedup divided by p, where p, the number of processors, is the ideal speedup. The traditional parallel efficiency is the efficiency in terms of speedup. Contrary to speedup, average speed is an indicator of uniprocessor efficiency, where uniprocessor efficiency is defined as average unit speed over peak uniprocessor speed. Maintaining average speed is equivalent to maintaining the uniprocessor efficiency. Under certain assumptions, maintaining average speed is also equivalent to maintaining the parallel efficiency [9]. However, in practice, these two approaches may lead to totally different results [9]. Unlike parallel efficiency, average speed does not inherit any deficiency of speedup. It does not require solving large problems on a single processor and does not give credit to slow computation, while parallel efficiency does.

Three different approaches have been proposed in [8] to obtain scalability.

1. The scalability can be *measured* using software by a control program that invokes the application program and searches for the run which has the desired fixed average unit speed.

2. The scalability can be *computed* by first finding the relation between average unit speed and execution time (or work) and then using equation (1) (or equation (4)).

3. The scalability can be *predicted* by deriving a general scalability formula.

The third approach, i.e. prediction, is the topic of this study. It is the simplest one among the three approaches, if a formula can be defined. A prediction formula is given in [8] for applications where communication cost is independent of problem size and work load is balanced among processors. By the definition of scalability (1), scalability can be predicted if and only if the scaled work size, W', can be predicted. Proposition 1 provides a way to obtain W'.

Proposition 1 *If parallel degradation exists, then for scalability (1)*

$$W' = \frac{ap'T_o}{1 - a\triangle}, \tag{2}$$

where a is the fixed average speed, $\triangle$ is the computing rate of a single processor, and T_o is the parallel processing overhead.

Proof: Since W' is the scaled work satisfying the isospeed requirement,

$$a = \frac{W'}{p'T_{p'}(W')}.$$

The parallel execution time, $T_{p'}(W')$, can be divided into two parts: ideal parallel processing time and parallel processing overhead, T_o.

$$T_{p'}(W') = \frac{T_1}{p'} + T_o = \frac{W'\triangle}{p'} + T_o,$$

where T_1 is the sequential execution time and T_1/p' is the ideal parallel execution time. Thus,

$$a = \frac{W'}{W' \triangle + T_o p'},$$

and

$$W' = \frac{ap'T_o}{1 - a\triangle}.$$

$\square$

Note that in Equation (2), a is the achieved average speed considering the parallel processing overhead, and $\triangle$ is the computing rate without considering the overhead. When parallel degradation does exist (i.e. $T_o > 0$), $\triangle^{-1} > a$ and, therefore, equation (2) is traceable. $T_o > 0$ is a necessary and sufficient condition of Proposition 1.

Combining scalability (1) and equation (2), we have

$$\psi(p, p') = \frac{W(1 - a\triangle)}{paT_o}. \tag{3}$$

Equation (3) is very useful. It not only gives a way to predict scalability, but more importantly, it shows the following three properties of isospeed scalability.

1. Scalability (1) increases with the decrease of the fixed average speed a.

2. $\triangle$, the computing rate of a single processor, is the inverse of single processor speed. Equation (3) shows that scalability increases with single processor speed.

3. Scalability increases with the decrease of degradation of parallelism T_o.

Property 1 is very reasonable. Scalability is the ability of a computing system to maintain performance when system size is scaled up. Property 1 shows that less effort is needed to maintain lower efficiency, if we consider $a\triangle$ as the uniprocessor efficiency. Equation (3) gives the relation between the effort (scalability) and performance (the fixed average speed) of an algorithm-machine combination. Property 1 also shows that, by adjusting the average speed a, isospeed scalability can be applied to a large class of algorithm-machine combinations, from massively parallel systems with relatively weak processing elements to supercomputers with a few powerful processors. Equation (3) also gives the relation between isospeed scalability, computing power of a single processor, and degradation of parallelism. Properties 2 and 3 show that isospeed scalability does not give credit to slow computing and communication. These two properties are very important in evaluation of computing systems. They distinguish isospeed scalability from parallel metrics based on speedup. It is known that speedup favors parallel systems with a high communication/computing ratio [9].

Although equation (3) is very useful, using it in performance prediction may not be as simple as it looks. The degradation of parallelism, T_o, which contains both communication and workload imbalance degradation, may be difficult to compute. Also, the single processor rate may vary with algorithm and problem size, especially for shared virtual memory machines [9]. A detailed case study is given in the next section to illustrate how the prediction formula could be used in practice, and how the predicted scalability could be used to evaluate machine architectures.

Finally, equation (4) shows how parallel execution time could be computed from scalability.

$$T_{p'}(W') = \psi^{-1}(p, p')T_p(W), \tag{4}$$

where $T_p(W)$, $T_{p'}(W')$ are the parallel execution times of solving the problem with the work of W and W' on a system of p and p' processors respectively. The computing rate of single processor, $\triangle$, is machine dependent. The degradation of parallelism, T_o, is both architecture and algorithm dependent. Equation (3) gives a way to find a good algorithm-machine combination in terms of scalability. Equation (4) shows larger scalability will lead to smaller execution time.

3 The Case Study

In this section, we discuss the case study for solving an application problem on KSR-1 parallel computers. We first give brief descriptions of the architecture and the application problem, and then present the measured performance and compare it with the predicted performance.

3.1 The Machine

Our case study was performed on the KSR-1 parallel computer. It has a distributed physical memory which makes a large ensemble size possible, and a shared address space which allows users to develop programs in a shared-memory-like environment.

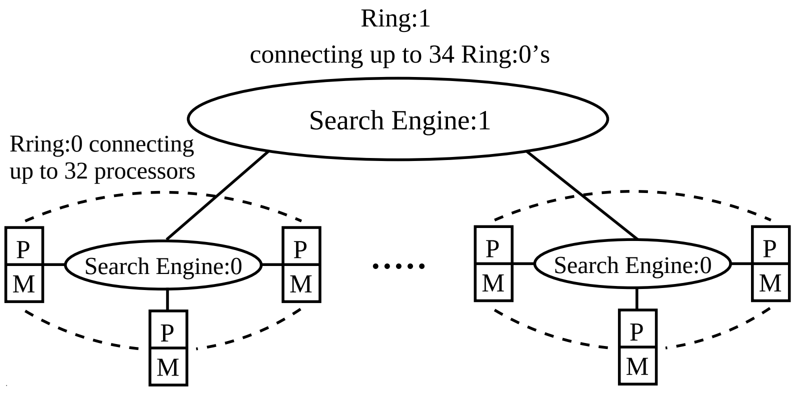
Ring:1
connecting up to 34 Ring:0's
Search Engine:1
Rring:0 connecting up to 32 processors
Search Engine:0
Search Engine:0
P
M
.....

Figure 1. Configuration of KSR-1 parallel computers.
p: processor M: 32 Mbytes of local memory

Figure 1 shows the architecture of the KSR-1 parallel computer. Each processor on the KSR-1 has 32 Mbytes of local memory. The CPU is a super-scalar processor with a peak performance of 40 Mflops in double precision. Processors are organized into different rings. The local ring (ring:0) can connect up to 32 processors, and a higher level ring of rings (ring:1) can contain up to 34 local rings with a maximum of 1088 processors.

Access to non-local data on KSR is provided by a hierarchy of Search Engines. The Search Engine SE:0 locates data in the local ring, while the Search Engine SE:1 provides data access between local rings. These different Search Engines are connected in a fat-tree-like structure. The memory hierarchy of KSR is shown in Figure 2.

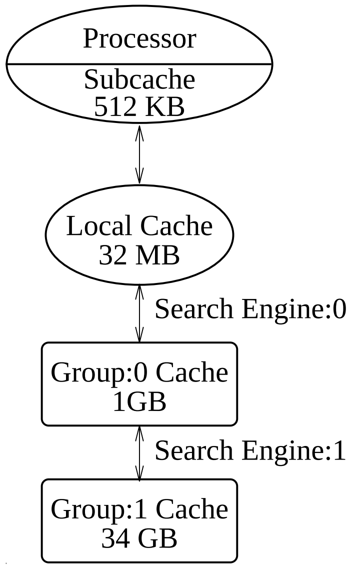

Figure 2. Memory hierarchy of KSR-1.

Each processor has 512 Kbytes of fast *subcache* which is similar to the normal cache on other parallel computers. This subcache is divided into two equal parts: an instruction subcache and a data subcache. The 32 Mbytes of local memory on each processor is called a *local cache*. A local ring (ring:0) with up to 32 processors can have 1 Gbytes total of local cache which is called the *Group:0 cache*. Access to the Group:0 cache is provided by Search Engine:0. Finally, a higher level ring of rings (ring:1) connects up to 34 local rings with 34 Gbytes of total local cache which is called *Group:1 cache*. Access to the Group:1 cache is provided by Search Engine:1. The entire memory hierarchy is called ALLCACHE memory by the Kendall Square Research. Access by a processor to the ALLCACHE memory system is accomplished by going through different Search Engines as shown in Figure 2. The latencies for different memory locations [4] are: 2 cycles for *subcache*, 20 cycles for *local cache*, 150 cycles for *Group:0 cache*, and 570 cycles for *Group:1 cache*.

3.2 The Application

The numerical algorithm used in this case study is the Householder Transformation algorithm for the QR factorization of matrices. It is used for solving the normal equation

$$\mathbf{A^T A x} = \mathbf{A^T b} \tag{5}$$

without explicitly forming $\mathbf{A^T A}$.

In many cases, for instance the inverse problem of partial differential equations [1], the normal equation system resulting from the discretization is too ill-conditioned to be solved directly. Tikhnov's regularization method [10] is frequently used in this case to increase numerical stability. The key step in solving the Regularized Least Squares Problem (RLSP) is to introduce a regularization factor $\alpha > 0$. Instead of solving (5) directly, we solve the following system

$$(\mathbf{A^T A} + \alpha\mathbf{I})\mathbf{x} = \mathbf{A^T b} \tag{6}$$

for $\mathbf{x}$. Equation (6) can also be written as

$$(A^T, \sqrt{\alpha}I)\begin{pmatrix} A \\ \sqrt{\alpha}I \end{pmatrix}\mathbf{x} = (A^T, \sqrt{\alpha}I)\begin{pmatrix} b \\ 0 \end{pmatrix} \tag{7}$$

or

$$B^T B\mathbf{x} = B^T \begin{pmatrix} b \\ 0 \end{pmatrix}, \tag{8}$$

so that the major task is to carry out the QR factorization for matrix B which has the structure

$$B = \left[\begin{array}{l} \boxed{\begin{array}{cccc} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1}^{(1)} & a_{m2}^{(1)} & \cdots & a_{mn}^{(1)} \\ \sqrt{\alpha} & & & \end{array}} \\ \qquad\quad \sqrt{\alpha} \\ \qquad\qquad\qquad \ddots \\ \qquad\qquad\qquad\qquad \sqrt{\alpha} \end{array}\right], \tag{9}$$

where we usually have $m \geq n$ with m of the same order as n. Matrix B is neither a complete full matrix nor a sparse matrix. The upper part is full and the lower part is sparse (in diagonal form). Because of the special structure in (9), not all elements in the matrix are affected in a particular transformation step. In the first step, all elements within the frame in matrix (9) will be affected. In each new step, the frame in (9) will shift downwards one row with the left most column out of the game. Therefore, at the ith step, the submatrix $\mathbf{B}_i$ affected in the transformation has the form:

$$B_i = \left[\begin{array}{cccc} a_{ii}^{(i)} & \cdots & \cdots & a_{in}^{(i)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m+i-1,i}^{(i)} & \cdots & \cdots & a_{m+i-1,n}^{(i)} \\ \sqrt{\alpha} & 0 & \cdots & 0 \end{array}\right]. \tag{10}$$

If the columns of matrix $\mathbf{B}_i$ of (10) are denoted by $\mathbf{b}_j^i$, i.e.

$$\mathbf{B}_i = [\mathbf{b}_i^i \ \mathbf{b}_{i+1}^i \cdots \mathbf{b}_n^i], \tag{11}$$

then the Householder Transformation can be described as:

```
Householder Transformation

Initialize matrix $B$
for $i = 1, n$
    1. $\alpha_i = -sign(a_{ii}^{(i)})(\mathbf{b}_i^{i^T} \mathbf{b}_i^i)^{1/2}$
    2. $\mathbf{w}_i = \mathbf{b}_i^i - \alpha_i \mathbf{e}_1$
    3. $\beta_j = \mathbf{w}_i^T \mathbf{b}_j^i (\alpha_i^2 - \alpha_i a_{ii}^{(i)}), \quad j = i+1, \cdots, n$
    4. $\mathbf{b}_j^i = \mathbf{b}_j^i - \beta_j \mathbf{w}_i, \quad j = i+1, \cdots n$
end for
```

The calculation of β_j's and updating of $\mathbf{b}_j^i$'s can be done in parallel for different indices j.

3.3 Scalability Analysis

Based on the definition of isospeed scalability, the work W' at processor number p' should keep the system ensemble running at the same average speed a as with p processors, so that

$$a = \frac{W}{pT_p(W)} = \frac{W'}{p'T_{p'}(W')}, \tag{12}$$

where $T_p(W)$ and $T_{p'}(W')$ are the execution times using p and p' processors respectively.

For the particular problem discussed here, the run time model is

$$T_p(n) = \left[\frac{2n^3}{p} + 3n^2\right]\tau + n^2\beta, \tag{13}$$

and the work is

$$W(n) = 2n^3 + 3n^2, \tag{14}$$

where n is the number of columns in a $2n \times n$ matrix to be transformed, p is the number of processors, τ is the rate of computing without communication overhead, and β is the latency for access of remote data in the Group:0 cache. We use τ, instead of $\triangle$, to represent the computing rate, because in practice the computing rate may vary with algorithm, problem size, and system size. We reserve the notation $\triangle$ for the theoretical computing rate. Following the discussion given in Section 2, the run time $T_p(n)$ in (13) can apparently be represented as

$$T_p(n) = T_C(n,p) + T_o(n,p), \tag{15}$$

where $T_C(n,p)$ is the computing time with ideal parallelism and $T_o(n,p)$ represents the degradation of parallelism. We then have

$$T_C(n,p) = \frac{2n^3 + 3n^2}{p}\tau,$$

$$T_o(n,p) = (3n^2 - \frac{3n^2}{p})\tau + n^2\beta.$$

The first term of T_o is due to the workload imbalance. The second term is due to the communication (remote memory access) delay. Using relation (2) we get

$$W' = \frac{ap'(-\frac{3n'^2}{p'}\tau + 3n'^2\tau + n'^2\beta)}{1 - a\tau}. \tag{16}$$

The matrix size n is the parameter used to adjust the problem size. Substituting

$$W' = 2n'^3 + 3n'^2$$

into (16), we have

$$2n'^3 + 3n'^2 = \frac{ap'(-\frac{3n'^2}{p'}\tau + 3n'^2\tau + n'^2\beta)}{1 - a\tau}$$

which eventually leads to

$$n' = \frac{3a\tau p' + a\beta p'}{2(1-a\tau)} - \frac{3}{2(1-a\tau)}. \tag{17}$$

Equation (17) is true for any work-processor pair which maintains the fixed average speed, assuming that τ and β are unchanged. In particular,

$$n = \frac{3a\tau p + a\beta p}{2(1-a\tau)} - \frac{3}{2(1-a\tau)}. \tag{18}$$

Combining equation (17) and (18), we have

$$(n' - n) = \frac{3a\tau + a\beta}{2(1-a\tau)}(p' - p), \tag{19}$$

which shows that the variation of n is in direct proportion to the variation of ensemble size, provided that τ and β are independent of the number of processors.

Equation (19) indicates that the matrix size n' must increase at the same rate as the number of processors p', to maintain the pre-specified average speed a. If $p' = mp$, then we will have $n' = mn$. Assuming n is large so that the cubical term in equation (14) is dominant, we have the relation

$$W'(n') = W'(mn) \approx m^3 W(n).$$

Therefore, the scalability of this algorithm-machine combination can be estimated as

$$\psi(p, p') = \psi(p, mp) \approx \frac{mp \cdot W}{pm^3 W} = \frac{1}{m^2}. \tag{20}$$

In particular, if $m = 2$, which means the number of processors is doubled for each case, the scalability will be approximately $\frac{1}{4}$.

It is clear from (19) that the parameters τ and β must first be determined before we can predict the execution time and scalability. With the run-time model given by (13), we can estimate τ and β in the model to fit the measured run times using the least squares method. Assume that the executions times $T_{p_1}(n_1), \cdots, T_{p_k}(n_k)$ are available on p_1, $p_2, \cdots, p_k$ processors, with problem sizes being n_1, $n_2 \cdots n_k$ respectively, we will have

$$\begin{aligned}\tau &= \frac{\sum_{i=1}^k b_i T_{p_i} \sum_{i=1}^k c_i^2 - \sum_{i=1}^k c_i T_{p_i} \sum_{i=1}^k b_i c_i}{\sum_{i=1}^k b_i^2 \sum_{i=1}^k c_i^2 - (\sum_{i=1}^k b_i c_i)^2} \\ \beta &= \frac{\sum_{i=1}^k b_i^2 \sum_{i=1}^k c_i T_{p_i} - \sum_{i=1}^k b_i c_i \sum_{i=1}^k b_i T_{p_i}}{\sum_{i=1}^k b_i^2 \sum_{i=1}^k c_i^2 - (\sum_{i=1}^k b_i c_i)^2}\end{aligned} \tag{21}$$

where

$$b_i = \frac{2n_i^3}{p_i} + 3n_i^2, \quad c_i = n_i^2.$$

4 Scalability Prediction and Its Application

The peak performance provided by vendors gives the hardware performance limit but can hardly be used to predict execution time accurately. For most application problems, the sustained speed is only a small percentage of the peak performance. The same argument applies to communication latency. The observed latency can be significantly different from the machine specifications. The architecture specification [4] for KSR-1 gives

$$\tau = 0.025 \quad (\mu s), \quad \beta_1 = 7.5 \quad (\mu s). \tag{22}$$

To determine the value of τ and β for this particular algorithm-machine pair, we ran the code on $p = 2$ and 4 processors and measured the total execution time $T_p(n)$ with $n = 362$ and $n = 512$ respectively. Then τ and β are calculated using the model in (21). The parameters obtained this way are

$$\tau' = 0.18 \quad (\mu s), \quad \beta' = 3.37 \quad (\mu s). \tag{23}$$

Comparing (22) and (23), we see that τ' is significantly larger than τ. The sustained computational speed is

$$\frac{1}{\tau'} = 5.56 \quad (Mflops)$$

which is about 14% of the peak performance of 40 Mflops. This speed includes all the effects of subcache misses and other overheads. On the other hand, the value of β' in (23) is significantly smaller than β of (22), which means the actual observed communication speed is faster. This can be attributed to two factoers:

1. Overlapping of communications with computations. In the Householder transformation, one processor calculates the pivoting column and then broadcasts it to all other processors. This broadcasting process can be partly overlapped with the other computations.

2. Automatic prefetch. The KSR-1 Fortran compiler analyzes loops and, whenever possible, generates instructions to prefetch remote data needed for subsequent loops, thus saving data

access time.

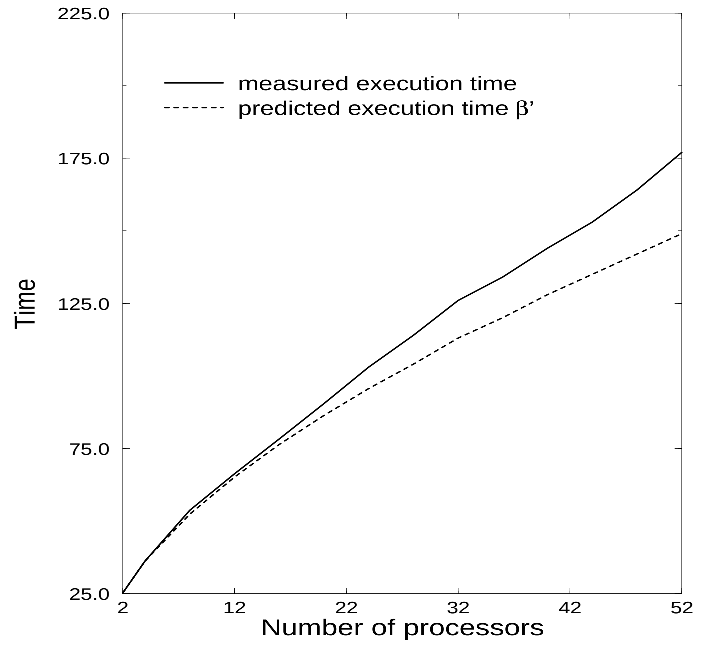

Figure 3. Measured and predicted execution time
Problem size is scaled up with available memory

Figure 3 shows both the measured execution time and the predicted execution time in seconds. The predicted execution time is based on equations (13) and (23). The problem size is scaled-up using the memory-bounded scale-up model [7], i.e. when the number of processors increases, the matrix size also increases to fill up the available local memory. For the RLSP application, memory requirement is a square function of the parameter n, and the computation count is a cubical function of n. That explains why the run time goes up with more processors.

It is clear from the figure that the predicted execution time matches the measured execution time well until $p = 22$. After that, the error increases significantly. This is due to the multi-ring structure of KSR-1. Each ring has 32 processors. Since several of the 32 processors are dedicated for I/O and control processes and are usually not used in computation, multi-ring communication is involved even for p less than (but close to) 32. This multi-ring communication requires data access to the Group:1 cache which slows the computations significantly. The listed access time for the Group:1 cache on KSR-1 is [4]

$$\beta_2 = 28.5 \quad (\mu s). \tag{24}$$

Again, the measured access time for our application is significantly different from the listed value, especially when most communications are within a single ring. To determine the communication delay for multiple rings, we ran the code on 36 processors and measured the execution time. Then the value of β was calculated from (13) by fixing $\tau = 0.18$ (μs) as given in (23). The new β value is

$$\beta'' = 6.27 \quad (\mu s) \tag{25}$$

which is about twice as large as that given in (23).

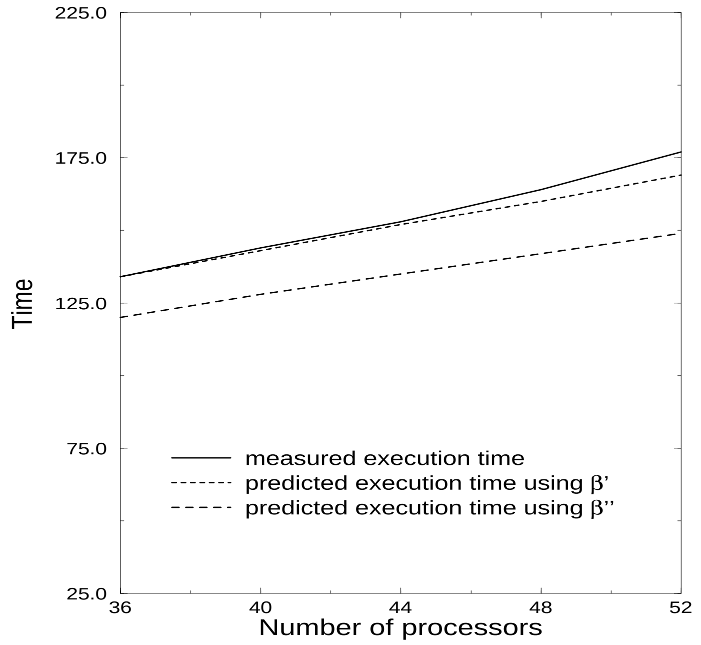

Figure 4. Measured execution time and predicted time using the adjusted parameters
Problem size is scaled up with available memory

Figure 4 shows the execution time for $p > 32$. We see that with the new value of β'', the predicted run time matches the measured execution time nicely.

Based on the test runs on $p = 2$, 4 and 36 processors and equation (17), the matrix size n' can be predicted. Table 1 shows the predicted and measured matrix sizes respectively. The average

size	1	2	4	8	16	32	56
predicted	–	54	115	238	484	976	2889
measured	29	57	109	230	461	1006	2773

Table 1. Predicted and measured matrix size

speed a maintained in this test is 3.25 Mflops, which is about 58% of the sustained speed in (23). From Table 1 we can see that the predicted matrix size is very close to the actual matrix size measured by running the code on 8, 16, 32, and 56 processors. The last column in Table 1 shows the predicted size n' using β''. If the β' given in (23) is used in predicting the matrix size, then n' will be 1715 at $p = 56$, which is significantly smaller than the measured n'. The difference shows

$\psi(p,p')$	1	2	4	8	16	32	56
1	1.00000	0.33238	0.07183	0.01652	0.00397	0.00097	0.00007
2		1.00000	0.21611	0.04971	0.01193	0.00292	0.00020
4			1.00000	0.23003	0.05520	0.01352	0.00092
8				1.00000	0.23999	0.05879	0.00398
16					1.00000	0.24499	0.01658
32						1.00000	0.06767
56							1.00000

Table 2. Predicted scalability of RLSP-KSR1 Combination

the influence of slower remote memory access of the Group:1 cache on scalability.

With the matrix sizes given in Table 1 and the parameters given in (23) and (25), we can compute the scalability $\psi(p,p')$. Table 2 and 3 give the predicted and measured scalability respectively. We can see that the predicted and measured scalabilities are fairly close. The prediction at ensemble size of 56 is based on the justified communication delay β''. Figure 5 depicts the difference between the measured scalability and the predicted scalability obtained by using β'. The curves in the figure represent measured and predicted $\psi(p,56)$ respectively with p varying from 1 to 56. Note that in order to see clearly the difference between the two curves in figure 5, we plotted $-\log(\psi(p,56))$, instead of $\psi(p,56)$. Therefore, the curve with lower $-\log(\psi(p,56))$ value actually represents higher scalability than the curve with higher $-\log(\psi(p,56))$ value.

A single bus is an efficient architecture to support the shared-memory communication model and has been used successfully in several commercial shared-memory machines. Due to network contention, the single bus architecture is difficult to use to support a large number of processors efficiently. All the commercially available machines with bus communication network share less than 40 processors. In order to build a scalable shared virtual memory machine, the architecture of KSR-1 is designed as a combination of buses and a fat-tree (see Section 3.1). Each local ring has 32 processors connected to a single bus. Then, the local rings are connected with the fat-tree-like structure. Theoretically, the computing system can be scaled up to any number of processors by increasing the number of levels of the tree. Figure 5 shows the limitation of the ring-tree approach. The scalability is severely reduced when inter-ring remote access is required. It shows that, unless inter-ring communication can be improved, uniprocessor efficiency will reduce quickly with the increase of ensemble size and high computing power may not be achievable by increasing the number of levels of the fat-tree.

The scalability difference given in figure 5 is based on the measured scalability and the measured τ and β'. Figure 6 shows the scalability difference with the theoretical performance data $\triangle$, β_1, and β_2, where the average speed is fixed at 58% of the peak performance. It gives the theoretical difference of the RLSP application when Group:1 communication is required. Comparing the

$\psi(p, p')$	1	2	4	8	16	32	56
1	1.00000	0.28382	0.08418	0.01830	0.00459	0.00089	0.00007
2		1.00000	0.29660	0.06446	0.01616	0.00313	0.00026
4			1.00000	0.21734	0.05449	0.01054	0.00088
8				1.00000	0.25070	0.04849	0.00406
16					1.00000	0.19343	0.01621
32						1.00000	0.08378
56							1.00000

Table 3. Measured Scalability of RLSP-KSR1 combination.

curves in figure 5 with those in figure 6, we can clearly see the similarity. Both figures show that the scalability with remote cache access is much lower than that without considering remote data access. The general trends in both figures are very similar. Since the curves in figure 6 were plotted based on machine specification, it shows that, while machine specification does not provide good estimate of execution time or speed, it does give a foundation to predict the influence of architecture variation on performance. Equation (3) is a useful tool to predict performance of an algorithm-machine pair, even when the computing system is scaled up from one level of architecture hierarchy to two levels. It gives the variation of performance even only the hardware specification is available. The influence of architecture variation is different on different algorithms. When architecture scales up from one level of hierarchy to another, an algorithm that performed worse than another algorithm at a less hierarchical architecture might become better on a more hierarchical architecture. The scalability formula (3) provides a guideline for chosing algorithms as system size is scaled up.

Figure 7 shows the scalability curves for the Givens Rotation algorithm [5], which can also be used to solved the least squares problem. The same machine specifications as those used for figure 6 are used in figure 7. We can see that the scalability of the Givens rotation algorithm is worse than that of the Householder algorithm. However, the difference is decreased when the system scales up. This demonstrates that the scalability of the Givens algorithm is less affected by the hierarchical remote cache access than the Householder algorithm is. The Givens algorithm may provide better scalability and, therefore, better execution time when the system size is large enough so that multi-level ring communication is required. Figure 6 and 7 show how algorithms could be compared with the notion of scalability.

The average speed a maintained in this study is about 58% of the sustained speed. The efficiency maintained is reasonably high. The scalability given in Table 2 and 3 could be higher if a was lower, as shown in equation (3). Also, the computing rate τ in general varies with the number of processors and problem size on any machine with memory hierarchy. For our implementation, since the initial problem size is large and it increases with the number of processors, the computing rate is quite

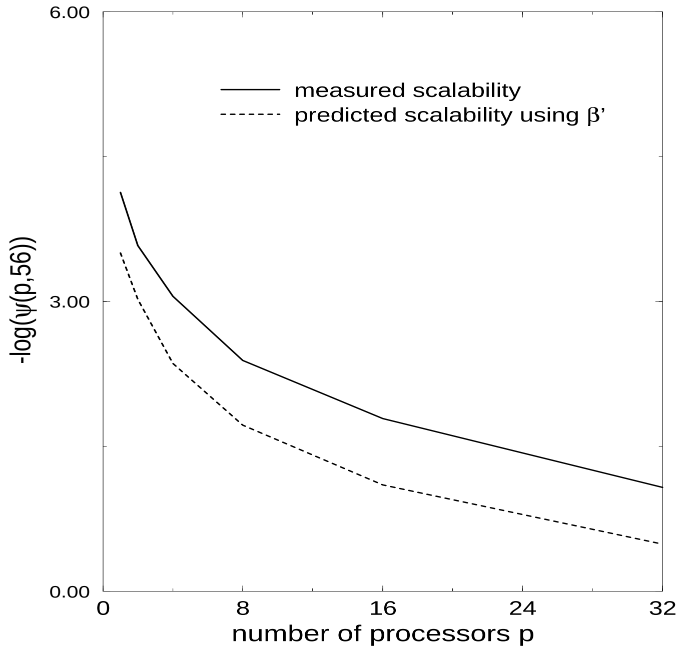

Figure 5. Measured and predicted scalability
Equation (23) is used in prediction

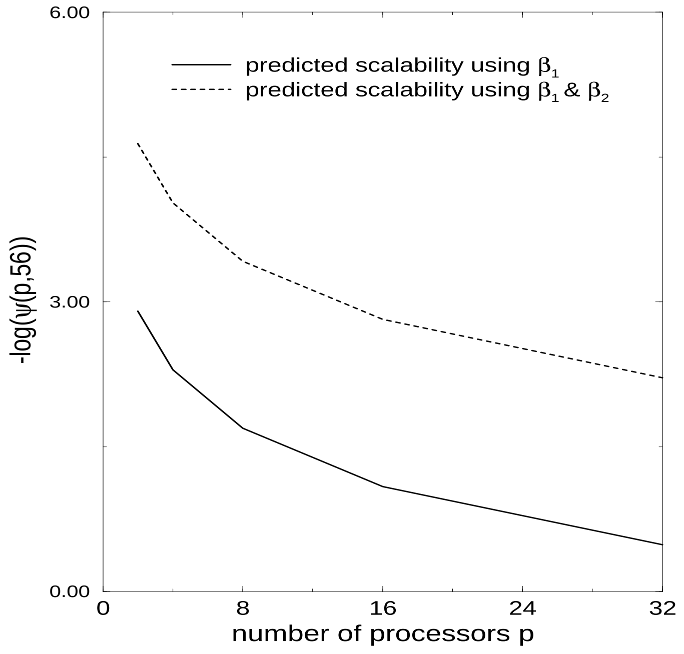

Figure 6. Predicted scalability using machine specifications

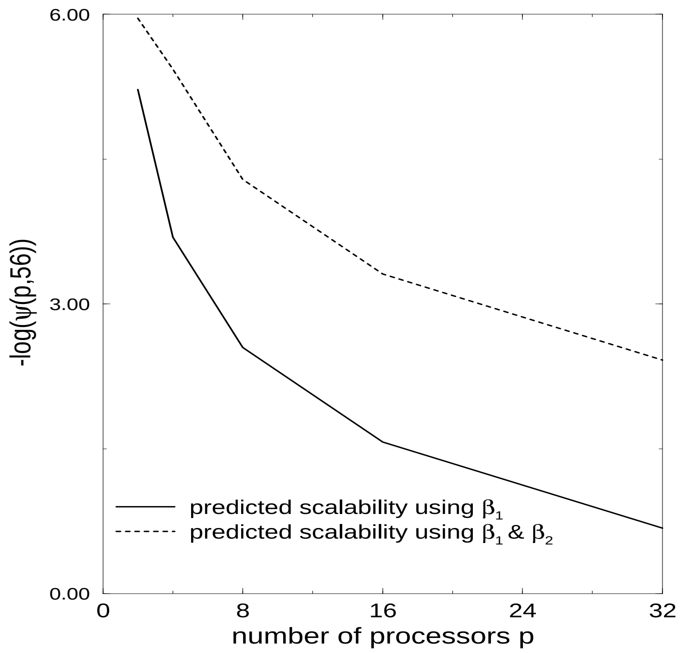

Figure 7. Predicted scalability of Givens rotation using machine specifications

stable. The scalability prediction will be more involved if the computing rate varies with the system size [6].

5 Conclusion

Recent trends in parallel processing suggest that the issue of performance prediction is becoming more complex and difficult. Massively parallel computing has been adopted as a cost-effective way to achieve high computing power. Sophisticated architectures have been proposed to deliver performance scalability with a large number of processors. Shared virtual memory and other kinds of system support, that hide the communication and other implementation details from the users, are becoming more prevalent. At the same time, with various architectures and algorithms available, performance prediction is becoming critical in of chosing an appropriate algorithm-machine pair for an application, especially when the machine has a sophisticated, hierarchical architecture. The study given in this paper is an attempt to combine simple formulas with run-time information to provide a reasonable prediction on modern parallel computers. A simple prediction formula is presented. Then, a case study is conducted on a multi-ring KSR-1 virtual memory machine to illustrate how the formula could be used in practice. Four different aspects are discussed in the paper. First, a method is proposed to measure the needed run-time parameters. Second, when the system size is scaled up from one level of architecture hierarchy to another level of hierarchy, an adjustment is proposed to catch the influence of the architecture variation. Experimental results on the multi-ring KSR-1 machine shows our predicted performance matchs the measured performance well, in both execution time and scalability. Then, with this case study, we have shown that it is possible to predict the influence of architecture hierarchy on scalability by simply using hardware specifications. Finally, we have discussed the issue of choosing an appropriate algorithm for a given application when the computing system is scaled up from one level of hierarchy to another.

Two basic problems have been addressed in this study: predicting the execution time and predicting the scalability. Like most existing models, the prediction of execution time relies on run-time information (such as τ and β) which may vary with problem and ensemble size. Our experiments show, however, that while hardware does not realize the advertized performance in solving actual applications, the relative performance of architectures and algorithms can be predicted and compared in terms of scalability given a hardware specification.

While the numerical experiment here was conducted on a KSR-1 machine, the result given in this study is not limited to the KSR-1 architecture. It is a general result of scalability prediction and should be useful in evaluation of any scalable architecture and algorithm.

Acknowledgment

The authors are grateful to the Cornell Theory Center for providing us access to the KSR parallel computer.

References

[1] Chen, Y. M., Zhu, J. P., Chen, W. H., and Wasserman, M. L. GPST inversion algorithm for history matching in 3-d 2-phase simulators. In *IMACS Trans. on Scientific Computing I* (1989), pp. 369–374.

[2] Grama, A. Y., Gupta, A., and Kumar, V. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE Parallel & Distributed Technoloty 1*, 3 (Aug. 1993), 12–21.

[3] Gustafson, J., Montry, G., and Benner, R. Development of parallel methods for a 1024-processor hypercube. *SIAM J. of Sci. and Stat. Computing 9*, 4 (July 1988), 609–638.

[4] Kendall Square Research. KSR technical summary. Waltham, USA, 1991.

[5] Pothen, A., and Raghavan, P. Distributed orthogonal factorization: Givens and Householder algorithms. *SIAM J. of Sci. and Stat. Computing 10* (1989), 1113–1135.

[6] Ramachandran, U., Shah, G., Ravikumar, S., and Muthukumarasamy, J. Scalability study of the KSR-1. Technical Report, GIT-CC 93/03, College of Computing, Georgia Institute of Technology, 1993.

[7] Sun, X.-H., and Ni, L. Scalable problems and memory-bounded speedup. *J. of Parallel and Distributed Computing 19* (Sept. 1993), 27–37.

[8] Sun, X.-H., and Rover, D. Scalability of parallel algorithm-machine combinations. *IEEE Transactions on Parallel and Distributed Systems* (June 1994), 599–613.

[9] Sun, X.-H., and Zhu, J. Shared virtual memory and generalized speedup. In *Proc. of the Eighth International Parallel Processing Symposium* (April 1994), pp. 637–643.

[10] Tikhnov, A. N., and Arsenin, V. *Solution of Ill-posed Problems.* John Wiley and Sons, 1977.

[11] Worley, P. T. The effect of time constraints on scaled speedup. *SIAM J. of Sci. and Stat. Computing 11*, 5 (Sept. 1990), 838–858.